Federated Query Processing: Challenges and Opportunities

Axel-Cyrille Ngonga Ngomo and Muhammad Saleem

Universität Leipzig, IFI/AKSW, PO 100920, D-04009 Leipzig
{lastname}@informatik.uni-leipzig.de

Abstract. The increasing numbers and volumes of RDF datasets are accompanied by increasingly complex information needs. Addressing such information needs commonly requires using federated queries, which are executed over several knowledge bases to compute a result set. The aim of this invited paper is to provide an overview of current challenges and opportunities in federated query processing. To this end, we summarize the results of recent state-of-the-art studies. We then derive potential cornerstones for future research in federated query processing.

1 Introduction

Like the Document Web, the Linked Data Web is characterized by a distributed architecture: *Data providers* make knowledge bases available, commonly as SPARQL endpoints. The data sources are often interlinked, which leads to information pertaining to particular resources being distributed across several knowledge bases. For example, information about presidents of the United States of America can be found in knowledge bases such as DBpedia and the New York Times (NYT). Consequently, computing the answer to certain user queries can demand compiling information from different knowledge bases. For example, gathering information about the presidents aforementioned (e.g., their political party and news sites pertaining to them) can require combining information from DBpedia and NYT. We call such queries *federated queries*.

Listing 1.1: Running example (LD3 from FedBench)

```
SELECT ?president ?party ?page
WHERE {
        ?president  rdf:type dbpedia:President .
        ?president dbpedia:nationality  dbpedia:United_States .
        ?president dbpedia:party ?party .
        ?x  nyt:topicPage ?page .
        ?x  owl:sameAs ?president .
}
```

While several types of federated queries exist, we focus on federated SPARQL queries herein. An example of such a query is shown in Listing 1.1. This query retrieves the political party and the NYT website of presidents who are nationals of the United States. The generic architecture of a federated SPARQL query engine is as shown in Figure 1. The input query q is first sent to a *parser/rewriter*, which reads and checks the query for whether it is a valid SPARQL query as well as rewrites it if necessary. The query is then forwarded to a *source selection* approach, which determines the subset of the knowledge bases $S_1, \ldots, S_n$ that must be queried to compute the result set of the input query. This is most commonly carried out by a triple-pattern-wise source selection, where the sources necessary to retrieve result sets for each triple pattern $T_1, \ldots, T_m$ of q are detected. After the computation and optimization of a query plan, the sources for each of the triple patterns are queried by means of a *federator*. The results are subsequently merged by means of the *integrator* and the query results are returned. Improving the performance of federated SPARQL query processing engines is equivalent to improving the performance of each of these components.

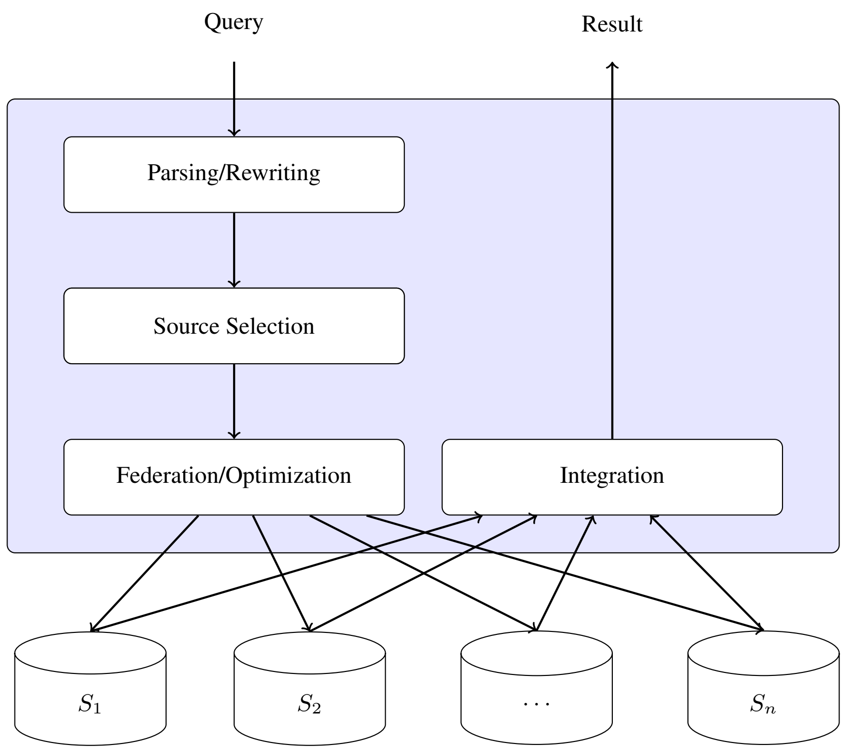

Fig. 1: Generic architecture of a SPARQL query federation engine

2 Challenges and Opportunities

This section addresses four of the most pressing challenges in the area of federated SPARQL query processing. We derive these challenges from [12,13]. In [12], we gathered query logs from SPARQL endpoints on the Web and performed an analysis of their structure and runtime. The queries from which the insights presented herein were derived can be found in LSQ's SPARQL endpoint.[1] Our other source, [13], presents a survey[2] and fine-grained evaluation of state-of-the-query federated query processing engines.

2.1 Source Selection

Source selection aims to map each of the triple patterns in the input query q to a set of data sources. Most SPARQL query federation approaches [4,5,11,19,20,16,15] perform a *triple-pattern-wise source selection* (TPWSS). Here, the goal is to identify the set of *relevant* (also called *capable*) sources for each individual triple patterns T_j of a query q, i.e., the set of sources S_i that return a non-empty result set when queried with this triple pattern. However, it is possible that a relevant source does not *contribute* to the final result set of a query. This is because the results from a particular data source can be excluded after performing *joins* with the results of other triple patterns contained in the same query. An overestimation of such sources increases the network traffic, the number of intermediate results and thus can significantly increase the overall query processing time [15]. *Join-aware TPWSS* strategies have been shown to yield great improvement potential [1,15]. For example, [15] can reduce the number of sources selection by approximately 47%. Still, there is a gap being between the optimal number of sources to select and the number of sources selected by current approaches. In particular, join-aware approaches based on authorities still fail to perform well on datasets where the same authorities are distributed across a large number of data sources. The development of better approaches for the characterization of data contained in single knowledge bases would further reduce both the network traffic and the source selection time.

In addition, our survey suggest that desirable features of federation systems are still to be addressed in literature. While approaches such as [16] allow approximating the recall of incomplete query results, source selection approaches dedicated to the computation of queries with `LIMIT` (approximately 17% of the queries in real query logs) remains work in progress. Such approaches could however improve the runtime of federated query processing further, as they would allow discarding sources that are capable but unnecessary to compute k results to the query. Similar insights can be derived for other SPARQL features.

2.2 Planning and Optimization

One of the most interesting results of the survey underlying [13] is that none of the systems surveyed combined cardinality estimations with join-aware source selection for planning. Devising an approach for computing more accurate join orderings can potentially yield improvements of several orders of magnitude on queries with large intermediate results. In addition, providing optimization of join implementations depending on the query can also lead to improved runtimes. Furthermore none of the current systems supports top-k queries according to the same study. However, approximately

[1] `http://lsq.aksw.org/sparql`
[2] The survey results are at `https://goo.gl/bAfJJM`.

17% of the queries available in the LSQ dumps use `LIMIT`. The provision of better planners that make use of information contained in SPARQL features is hence yet another opportunity for the development of more efficient federation systems. Static planning approaches could derive more efficient plans by having to deal with less sources.

An area of research which has been paid little attention to within the setting of federated queries is dynamic planning [1]. During the execution of federated queries, the planner most commonly generates a plan, which is executed as is. Therewith, current federation engines fail to make use of intermediary information they receive while executing portions of the queries. For example, receiving small intermediary results where large intermediary results were expected is a potent hint towards the subsequent steps of the plan needing readjustment. New results on dynamic planning in other areas of research such as link discovery [9] suggest that runtimes can be reduced by more than 80%. Approaches which can update their cost functions and therewith also their query processing plan should lead to a new generation of SPARQL federation engines.

2.3 Integration

The role of the integration module is to gather result sets from the previous step and to merge these result sets to the final result of the engine. Caching seems to be an obvious solution to improving the runtime of federated queries. In particular, tailored caching solutions have been shown to outperform generic caching approaches significantly (see, e.g., [7,10]). However, non of the systems we surveyed rely on caching the intermediate results when processing federated SPARQL queries. Thus, they were unable to reuse information on highly recurrent queries. Providing novel caching methods for the aggregation of results is thus one low-hanging fruit for future research. Another improvement would be in the development of polymorphic implementation of joins. Being able to choose the correct join implementation for merging the results gathered from a set of SPARQL endpoints can potentially lead to further improvements of the overall runtime of federated queries or even of triple stores.

2.4 Benchmarking

The main question that remains after composing a federated system is "how well does the new approach perform"? A large number of benchmarks for SPARQL have been devised over the last years [2,3,6,8,17,18,14]. However, only a few tackle federated query processing. While LargeRDFBench[3] is being used in a growing number of experiments, the most popular federated benchmark is FedBench [17]. This benchmark consists of 9 datasets and 25 queries. A deeper look into this benchmark however suggests that the queries it provised do not reflect real queries as found in query logs. For example, none of FedBench queries uses `ORDER BY`, `DISTINCT`, `REGEX` or `LIMIT`. However, these features are common in real queries. In addition, the average runtime of these queries (approx. 2 s for [15]) makes the evaluation of the significance of runtime differences between systems tedious to evaluate. We thus suggest that the queries in this benchmark are not sufficient to carry out a thorough evaluation of the performance of existing

[3] `https://github.com/AKSW/LargeRDFBench`

systems, especially when aiming to have an idea of their performance in real settings. A look at the size of the datasets included also suggest that the benchmark has stopped reflecting the complexity of the Data Web in which we currently live. The results of the evaluation in [13] suggest that benchmarks for federated queries must:

1. *SPARQL features*. While FedBench queries do not use a large number of SPARQL features (e.g.., `LIMIT`), LSQ clearly show that most SPARQL features are often used in Web queries (e.g., `LIMIT` is used in approximately 17% of the LSQ queries). We'd suggest including more queries into existing benchmarks that allow evaluating the performance of existing systems for all SPARQL features.
2. *Large data sources*. A large number of the current data sources dispatch billions of RDF triples. According to LodStats,[4] 144 SPARQL endpoints contain an average of 1.03 billion triples. Reflecting these ever growing dataset sizes is of central importance to implement the vision of realistic benchmarks that allow selecting the right solution for a given problem.
3. *Duplicated data*. The distributed and decentral architecture of the Linked Data Web leads to (1) data sources being duplicated or replicated (e.g., DBLP) and (2) portions of datasets being duplicated across different endpoints (e.g., copies of DBpedia). Modern federated benchmark must contain this characteristic of the Linked Data Web to allow evaluating the performance of existing systems. That the presence of duplicates can lead to different rankings in performance is shown in [13].
4. *Complex queries*. The largest query available in the LSQ dataset encompasses 37 triple patters. While the distribution of the number of triple patterns is a long-tailed distribution, this piece of information suggests that modern federation engines can be confronted with very large SPARQL queries. Including such queries into existing benchmarks would ensure a better and more realistic assessement of SPARQL federation systems. With this requirement also comes the needs for the inclusion of queries with *large intermediary results* and *large result sets*.

3 Conclusion

The aim of this paper was to give an overview of federated SPARQL query federation. In addition, we aimed to present some of the challenges and opportunities linked to this area of research. We presented a generic architecture for federation systems and derived challenges for the modules underlying these systems. We also addressed the current state of benchmarking federated RDF triple stores and

4 Acknowledgment

This work was supported by the project HOBBIT, which has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 688227.

[4] `http://stats.lod2.eu/`

References

1. M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. ANAPSID: an adaptive query processing engine for SPARQL endpoints. In *ISWC*, 2011.
2. G. Aluç, O. Hartig, M. T. Özsu, and K. Daudjee. Diversified stress testing of rdf data management systems. In *ISWC*. 2014.
3. C. Bizer and A. Schultz. The berlin sparql benchmark. *IJSWIS*, 5(2):1–24, 2009.
4. A. Charalambidis, A. Troumpoukis, and S. Konstantopoulos. Semagrow: Optimizing federated sparql queries. In *SEMANTICS*, 2015.
5. O. Görlitz and S. Staab. Splendid: Sparql endpoint federation exploiting void descriptions. In *COLD at ISWC*, 2011.
6. Y. Guo and J. Heflin. LUBM: A benchmark for owl knowledge base systems. *JWS*, 2005.
7. M. Martin, J. Unbehauen, and S. Auer. Improving the performance of semantic web applications with sparql query caching. In *The Semantic Web: Research and Applications*, pages 304–318. Springer, 2010.
8. M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo. Dbpedia sparql benchmark - performance assessment with real queries on real data. In *International Semantic Web Conference*, pages 454–469, 2011.
9. A.-C. Ngonga Ngomo. Helios–execution optimization for link discovery. In *The Semantic Web–ISWC 2014*, pages 17–32. Springer, 2014.
10. A.-C. Ngonga Ngomo and M. Hassan. The lazy traveling salesman – memory management for large-scale link discovery. In *Extended Semantic Web Conference*. Springer, 2016.
11. B. Quilitz and U. Leser. Querying distributed rdf data sources with sparql. In *Extended Semantic Web Conference*, 2008.
12. M. Saleem, M. I. Ali, A. Hogan, Q. Mehmood, and A.-C. Ngonga Ngomo. Lsq: The linked sparql queries dataset. In *The Semantic Web-ISWC 2015*, pages 261–269. Springer, 2015.
13. M. Saleem, Y. Khan, A. Hasnain, I. Ermilov, and A.-C. Ngonga Ngomo. A fine-grained evaluation of sparql endpoint federation systems. *Semantic Web*, (Preprint):1–26, 2015.
14. M. Saleem, Q. Mehmood, and A.-C. Ngonga Ngomo. FEASIBLE: A featured-based sparql benchmark generation framework. In *ISWC*, 2015.
15. M. Saleem and A.-C. Ngonga Ngomo. HiBISCuS: Hypergraph-based source selection for sparql endpoint federation. In *Extended Semantic Web Conference*, 2014.
16. M. Saleem, A.-C. Ngonga Ngomo, J. X. Parreira, H. F. Deus, and M. Hauswirth. Daw: Duplicate-aware federated query processing over the web of data. In *ISWC*, 2013.
17. M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. Fedbench: a benchmark suite for federated semantic data query processing. In *ISWC*, 2011.
18. M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. Spˆ 2bench: a sparql performance benchmark. In *ICDE*, pages 222–233, 2009.
19. A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *ISWC*, 2011.
20. X. Wang, T. Tiropanis, and H. C. Davis. Lhd: Optimising linked data query processing using parallelisation. In *LDOW at WWW*, 2013.